

Ramblr

Making Reassembly Great Again

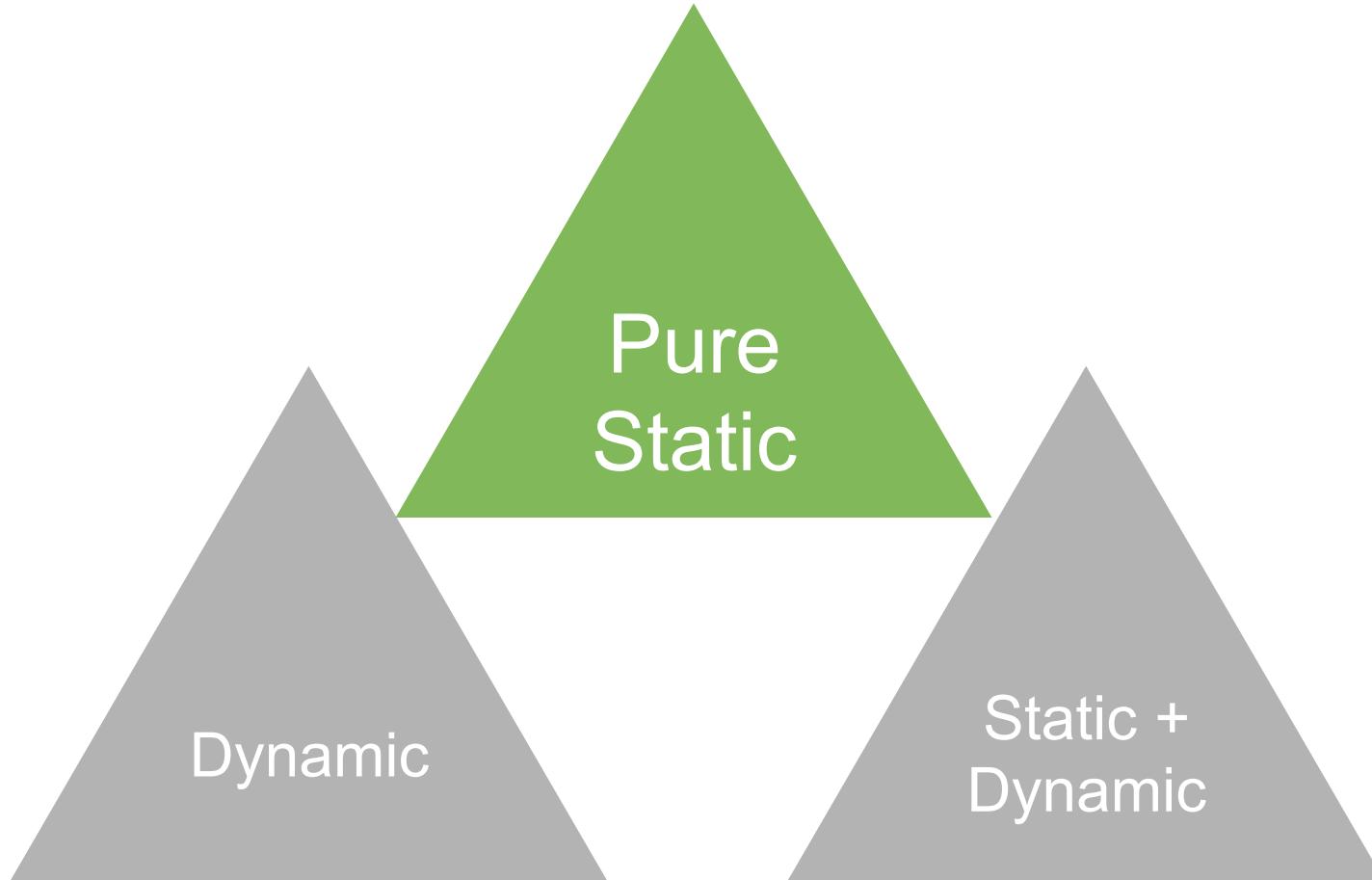
Ruoyu “Fish” Wang, Yan Shoshitaishvili, Antonio Bianchi,
Aravind Machiry, John Grosen, Paul Grosen,
Christopher Kruegel, Giovanni Vigna



Motivation



Available Solutions



What is Binary Reassembly?



Disassembl
e

	.text
400100	mov [6000a0], eax
400105	jmp 0x40020d
...	
40020d	mov [6000a4], 1
	.data
6000a0	.long 0xc0deb4be
6000a4	.long 0x0



Disassembl
e

	.text
target	mov [data_0], eax
	jmp target
	...
	mov [data_1], 1
	.data
data_0	.long 0xc0deb4be
data_1	.long 0x0



Patch &
Assemble

.text	
400100	mov [6000a0], eax
400105	jmp 40020d
40020d	CRASH!
40020f	mov [6000a4], 1
.data	
6000a0	"cat\x00"
6000a4	.long 0x0
6000a8	

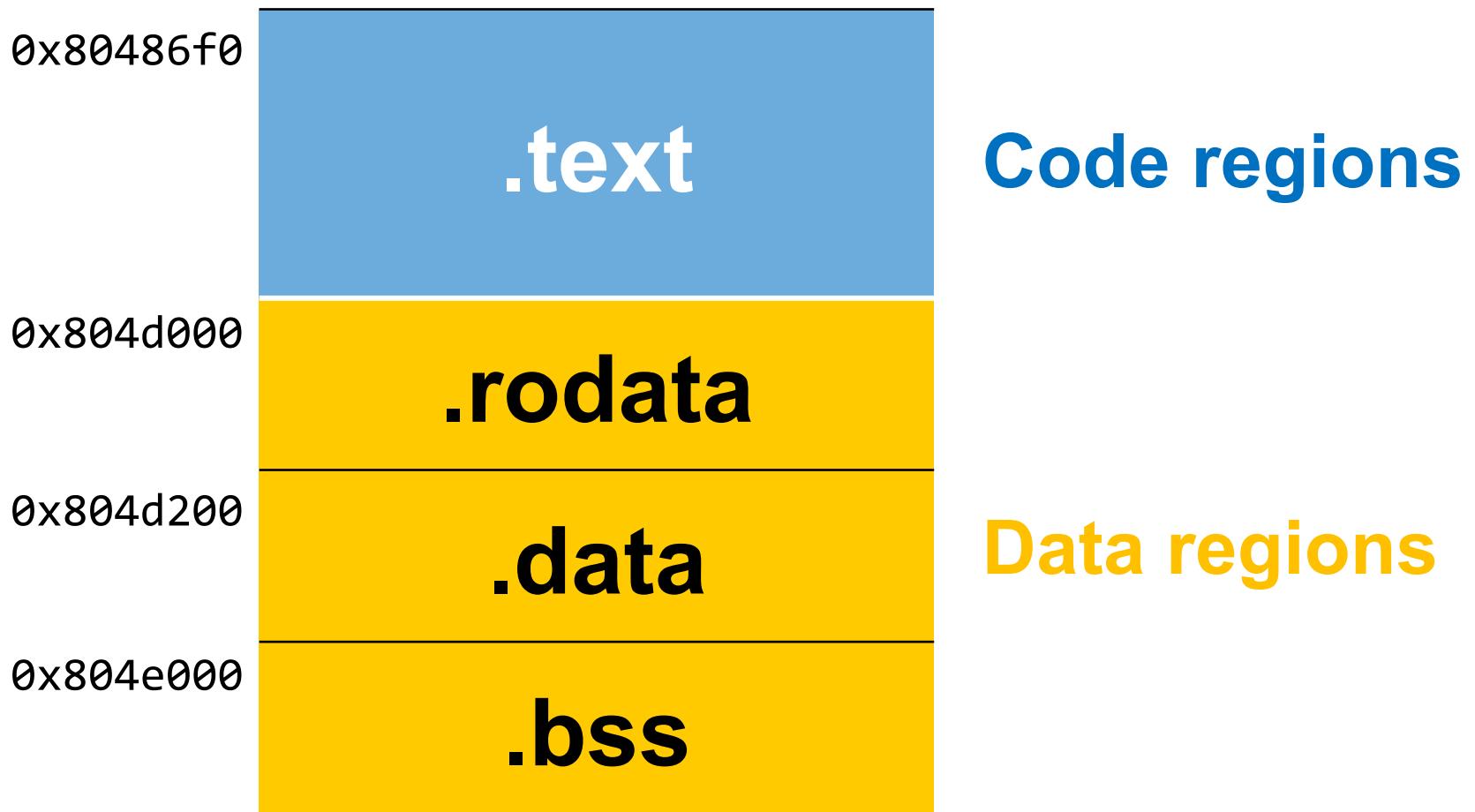
Non-relocatable Assembly



Patch &
Assemble

	.text
	mov [data_0], eax
	jmp target
	...
	mov [CRASH1], 1
target	mov [data_1], 1
	.data
data_0	.long 0xc0deb4be
data_1	“tagx000”
data_0	.long 0xc0deb4be
data_1	.long 0x0

Relocatable Assembly



0x80486f0

.text

0x804d000

.rodata

0x804d200

.data

0x804e000

.bss

...

.data:

804d538:

804d53c:

804d540:

```
push    ebp
mov     ebp, esp
sub     esp, 0x48
mov     DWORD PTR [ebp-0x10], 0x0
mov     DWORD PTR [ebp-0xc], 0x0
mov     DWORD PTR [ebp-0xc], 0x80540a0
mov     eax, 0xfb7
mov     WORD PTR [ebp-0x10], ax
mov     eax, ds:0x805be60
test   eax, eax
jne    0x804895b
mov     eax, ds:0x805be5c
```

0x8048eec
0x8048f05
0x8048f1e

Uroboros

USENIX Sec '15

Problems

HEY, THIS IS A VALUE,
NOT A POINTER!



False Positives

MAN, THIS IS ABSOLUTELY A
POINTER. WHY CAN'T YOU TELL?



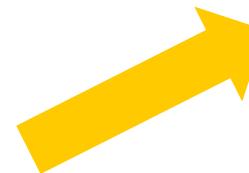
False Negatives

False Positives

Problem: Value Collisions

```
/* stored at 0x8060080 */
static float a = 4e-34;
```

A Floating-point Variable a



8060080	.db 3d
8060081	.db ec
8060082	.db 04
8060083	.db 08

Byte Representation



8060080	label_804ec3d
---------	---------------

Interpreted as a Pointer

False Negatives

Problem: Compiler Optimization

```
int ctrs[2] = {0};

int main()
{
    int input = getchar();
    switch (input - 'A')
    {
        case 0:
            ctrs[input - 'A']++;
            break;
        ...
    }
}
```

A code snippet allows **constant folding**

False Negatives

Problem: Compiler Optimization

```
int ct  
int ma  
{  
in  
sw  
{
```

$$0x804a034 - 'A' * \text{sizeof}(int) = 0x8049f30$$

```
a034  
], 1
```

not

A code snippet allows **constant folding**

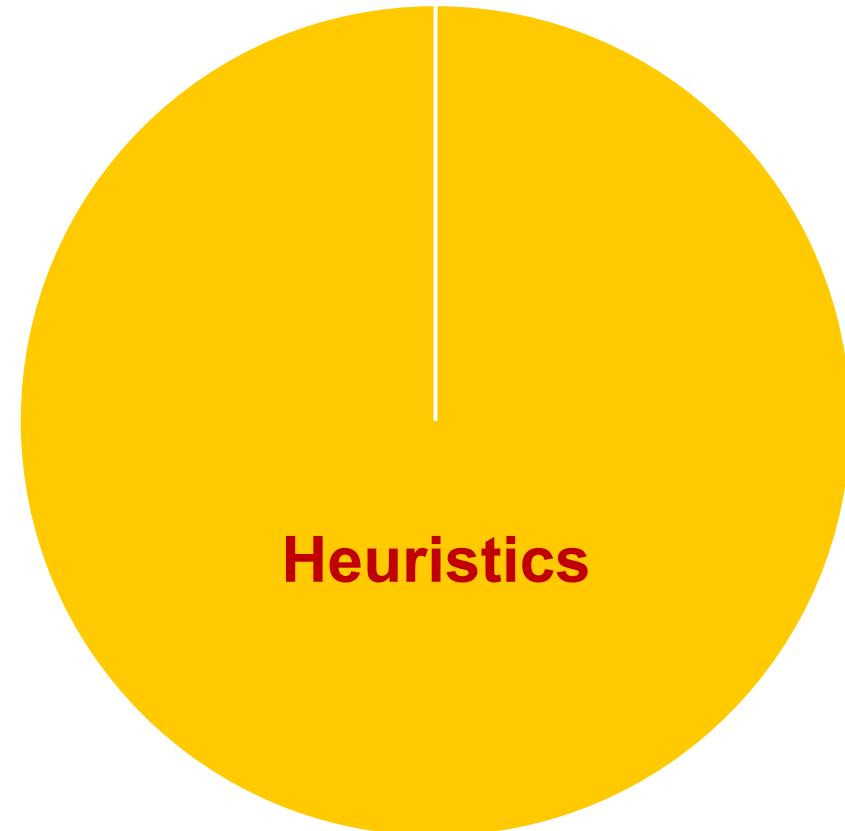
Compiled in Clang with -O1

Our Approach

Naïve Strategy

False Positives

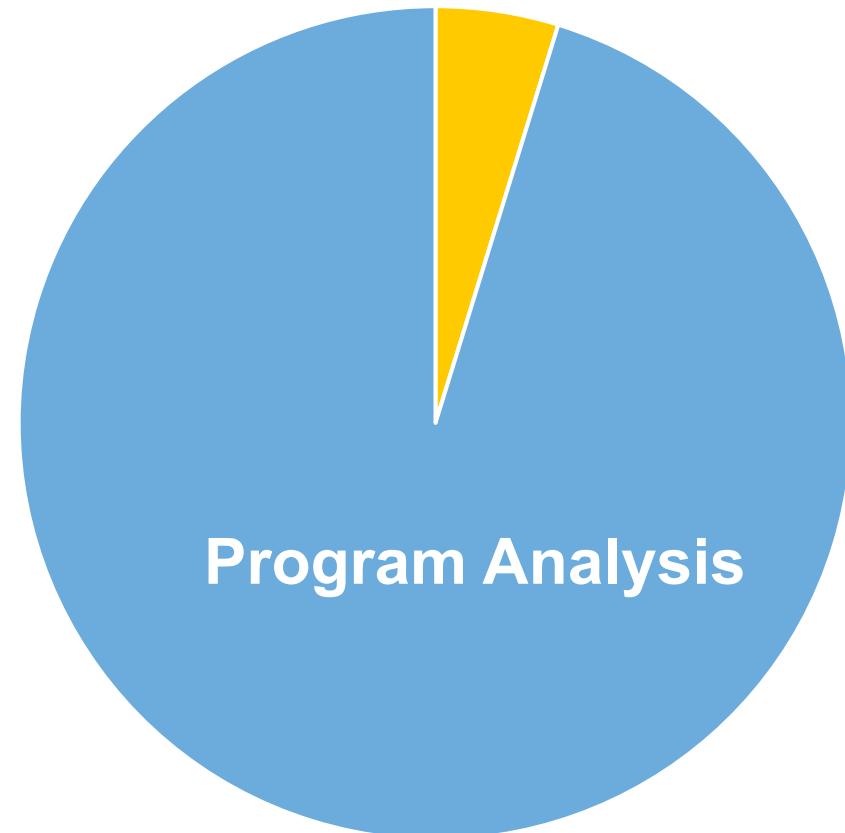
False Negatives



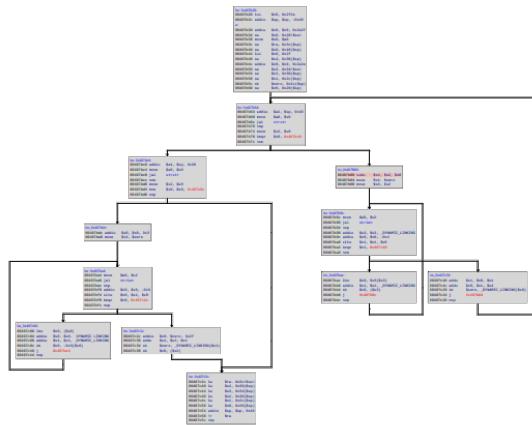
Ramblr

Heuristics

- False Positives**
- False Negatives**



Pipeline



CFG
Recovery

0x804850b	Pointer
0xa	Integer
0xdc5	Integer
63 61 74 00	String
0x80484a2	Pointer
0x804840b	Pointer
0xa0000	Integer

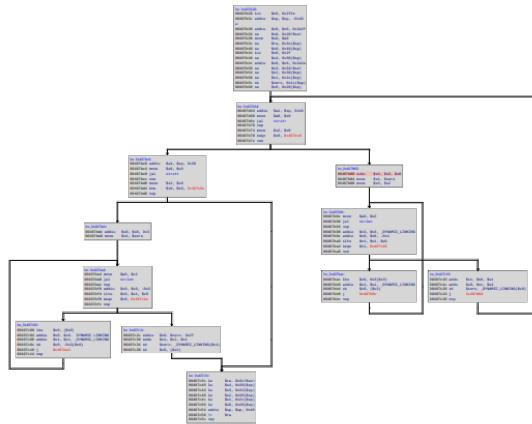
Content Classification

```
push    offset label_34
push    offset label_35
cmp     eax, ecx
jne    label_42

.label_42:
mov     eax, 0x12fa9e5
...
```

Symbolization
&
Reassembly

Pipeline



CFG
Recovery

Content Classification

0x804850b	Pointer
0xa	Integer
0xdc5	Integer
63 61 74 00	String
0x80484a2	Pointer
0x804840b	Pointer
0xa0000	Integer

```
push    offset label_34
push    offset label_35
cmp     eax, ecx
jne    label_42

.label_42:
mov     eax, 0x12fa9e5
...
```

Symbolization
&
Reassembly

CFG Recovery



31 ed 5e 89
e1 83 e4 f0
50 54 52 68
00 25 05 08

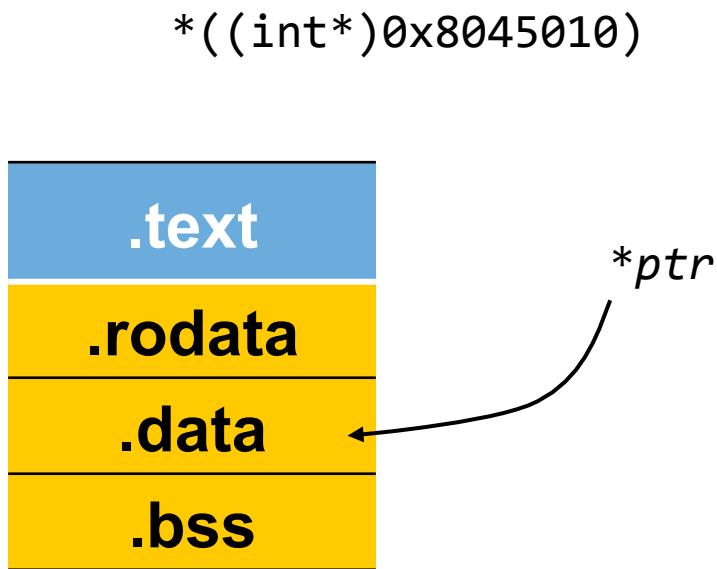
0x80486f0:
xor ebp, ebp
pop esi
mov ecx, esp
and esp, 0xffffffff0
push eax
push esp
push edx
...

Recursive Disassembly

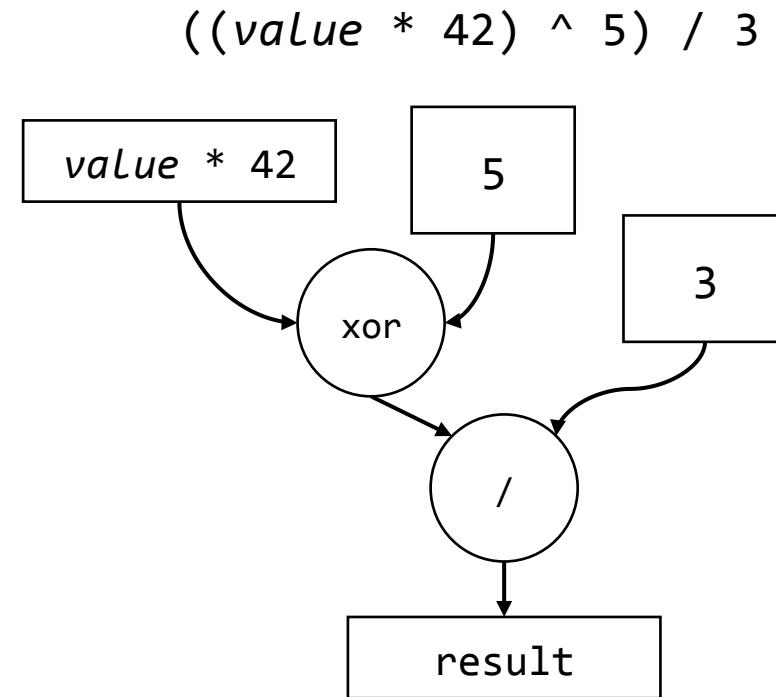


Iterative Refinement

Content Classification



A Typical Pointer



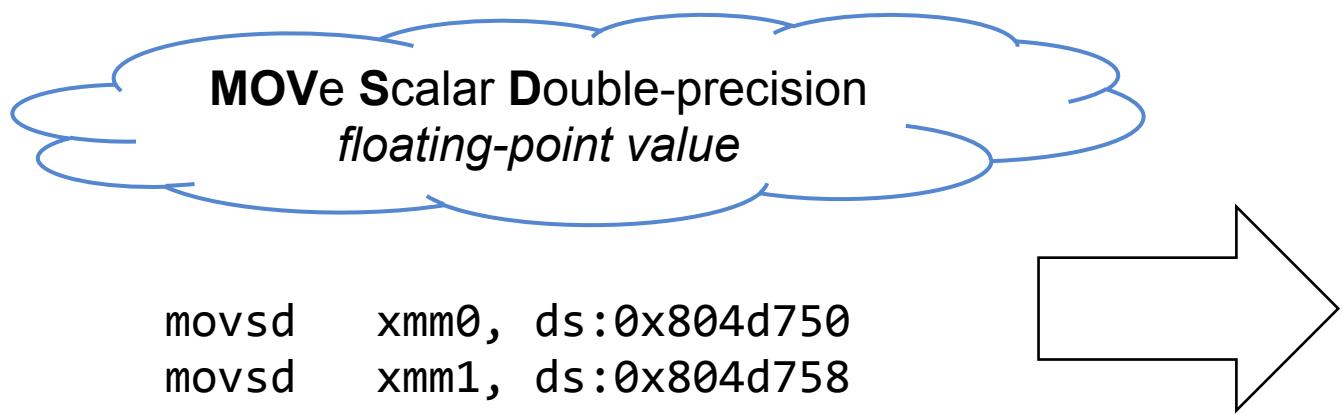
A Typical Value

Content Classification

Type Category	Examples
Primitive types	Pointers, shorts, DWORDs, QWORDs, Floating-point values, etc.
Strings	Null-terminated ASCII strings, Null-terminated UTF-16 strings
Jump tables	A list of jump targets
Arrays of primitive types	An array of pointers, a sequence of integers

Data Types that Ramblr Recognizes

Content Classification



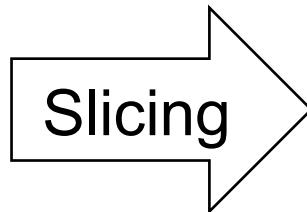
Two floating-points

804d750 Floating point integer
804d758 Floating point integer

Recognizing Types during CFG Recovery

Content Classification

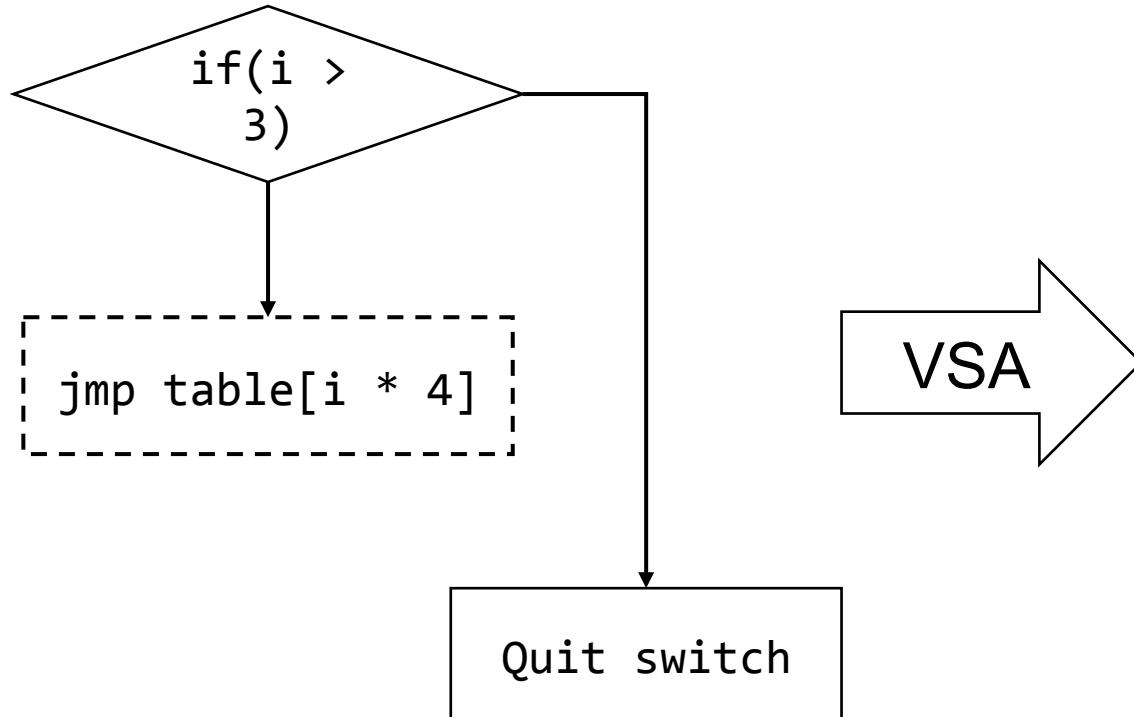
```
chr = _getch();
switch (i)
{
    case 1:
        a += 2; break;
    case 2:
        b += 4; break;
    case 3:
        c += 6; break;
    default:
        a = 0; break;
}
```



```
switch (i)
{
    case 1:
        ...
    case 2:
        ...
    case 3:
        ...
    default:
        ...
}
```

Recognizing Types with Slicing & VSA

Content Classification



$i = [0, 2]$ with a stride of 1

A jump table of 3 entries

table[0]	Pointer, jump target
table[1]	Pointer, jump target
table[2]	Pointer, jump target

Recognizing Types with Slicing & VSA

False Negatives

Base Pointer Reatribution

```
int ctrs[2] = {0};

int main()
{
    int input = getchar();
    switch (input - 'A')
    {
        case 0:
            ctrs[input - 'A']++;
            break;
        ...
    }
}
```

```
; Assuming ctrss is stored at 0x804a034
; eax holds the input character
; ctrss[input - 'A']++;
add    0x8049f30[eax * 4], 1
...
.bss
804a034: ctrs[0]
804a038: ctrs[1]
```

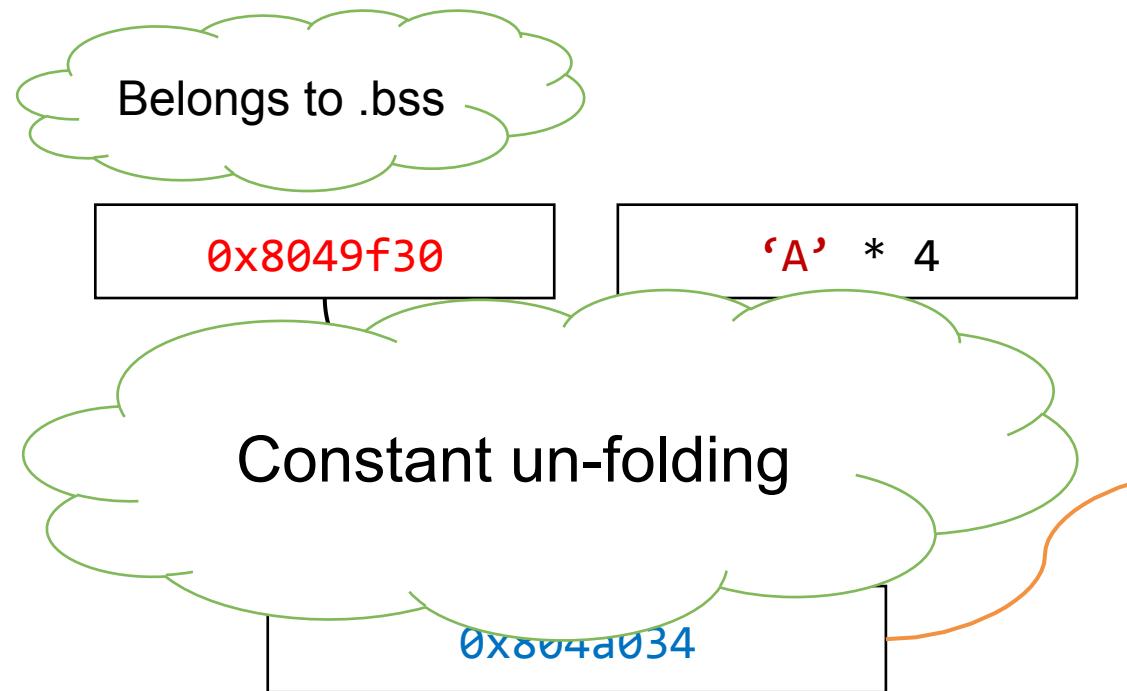
A code snippet allows **constant folding**

Compiled in Clang with `-O1`



False Negatives

Base Pointer Reattribution



The Slicing Result

```
; Assuming ctrs is stored at 0x804a034  
; eax holds the input character  
; ctrs[input - 'A']++;  
add    0x8049f30[eax * 4], 1
```

...

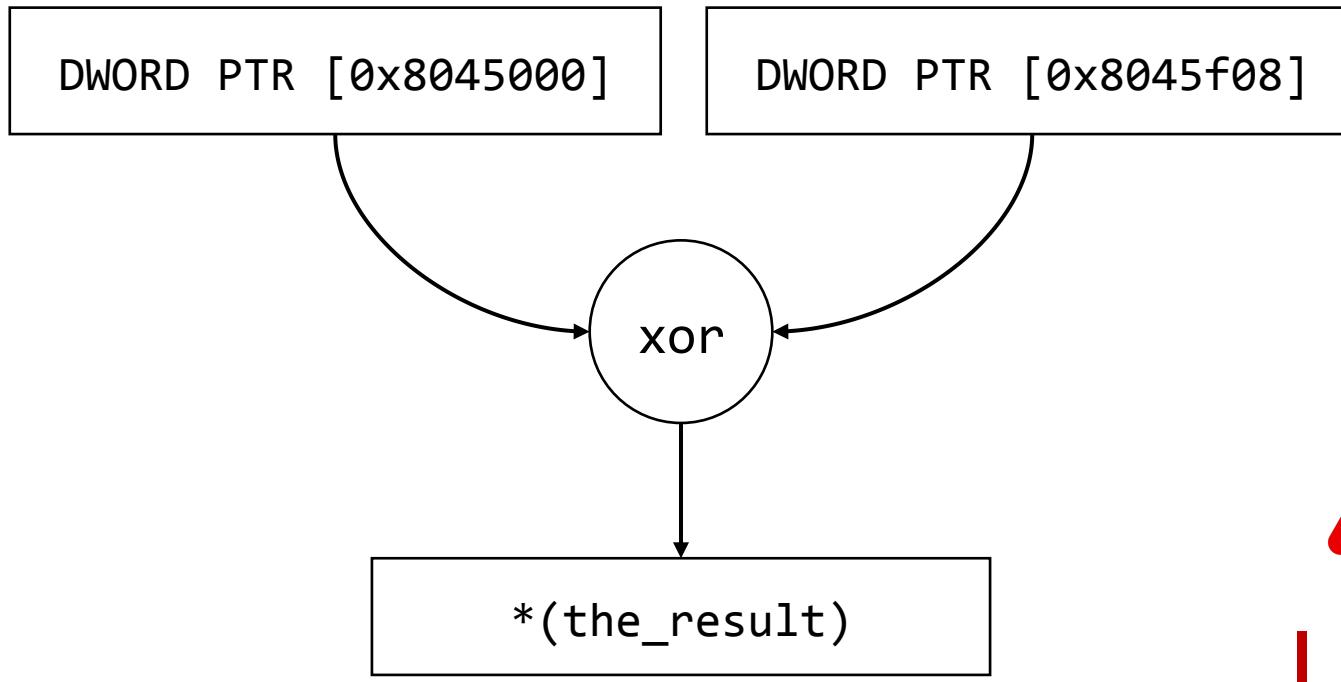
.bss

```
804a034: ctrs[0]  
804a038: ctrs[1]
```

Compiled in Clang with -O1

0x8049f30 does not belong to any section

Safety Heuristics: Data Consumer Check



I GIVE UP

Unusual Behaviors Triggering the Opt-out Rule

Symbolization & Reassembly

0x400010	→	label_34
0x400020	→	label_35
0x400a14	→	label_42
...		
0x406000	→	data_3

Symbolization

```
push    offset label_34
push    offset label_35
cmp     eax, ecx
jne    label_42

.label_42:
mov     eax, 0x12fa9e5
...
```

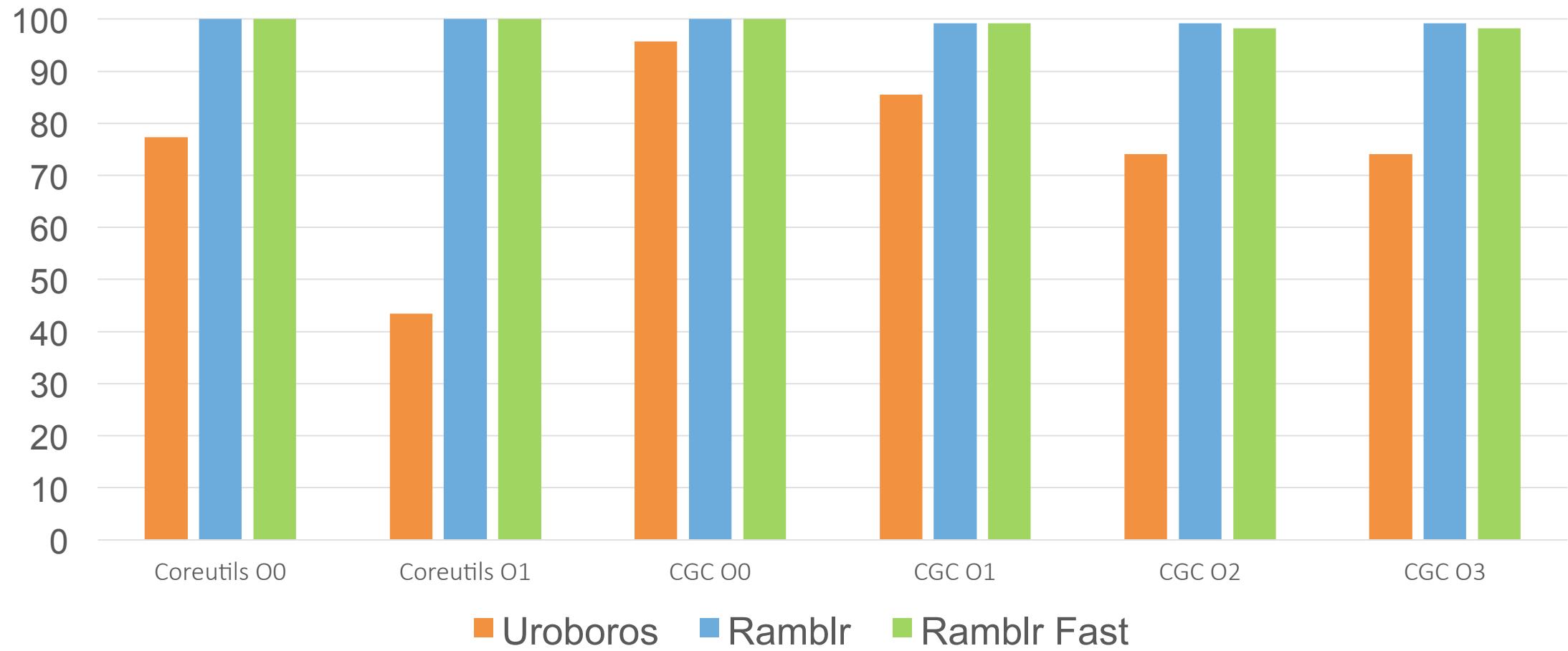
Assembly Generation

Evaluation

Data sets

	Coreutils 8.25.55	Binaries from CGC
Programs	106	143
Compiler	CGC 5	Clang 4.4
Optimization levels	O0/O1/O2/O3/Os/Ofast	
Architectures	X86/AMD64	X86
Test cases	Yes	Yes
Total binaries	1272	725

Brief Results: Success Rate





A large, stylized logo for "SHELLPHISH" is displayed. The word is written in a bold, black, sans-serif font. The letter "S" is designed to look like a shark's head, with a white eye and mouth area. The "H"s have vertical fins extending upwards from their top loops. The "P" and "L"s have vertical fins extending downwards from their bottom loops. The "I"s are small vertical lines, and the "H" at the end is a standard shape.



Ramblr is the foundation of ...

- Patching Vulnerabilities
- Obfuscating Control Flows
- Optimizing Binaries
- Hardening Binaries

SHELLPHISH

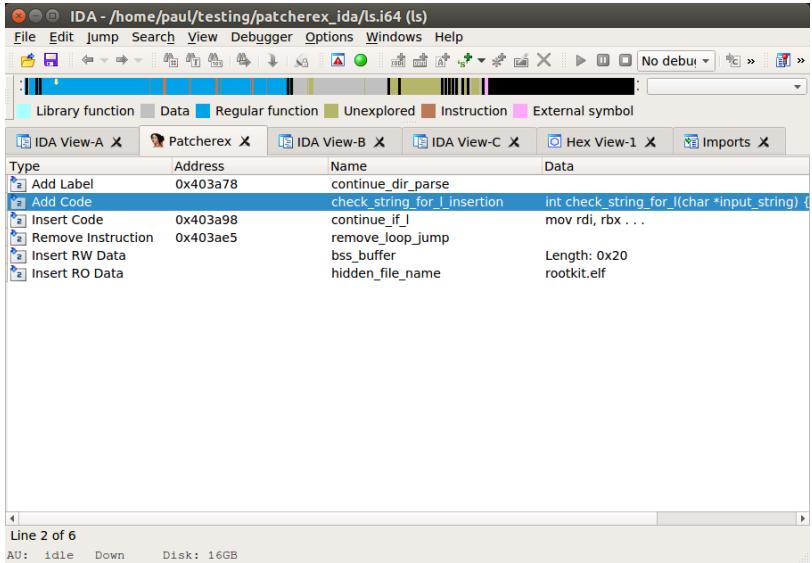


Conclusion

Conclusion

- Identified challenges in reassembling
 - Proposed a novel composition of static analysis techniques
 - Developed a systematic approach to reassemble stripped binaries
-
- ✓ Ramblr is open-sourced
 - ✓ Extra data-sets and usable tools will be released soon

Tools



RamblIr IDA Plugin



Patcherex

The screenshot shows the angr Management interface with a disassembly view. The left pane lists functions with their addresses:

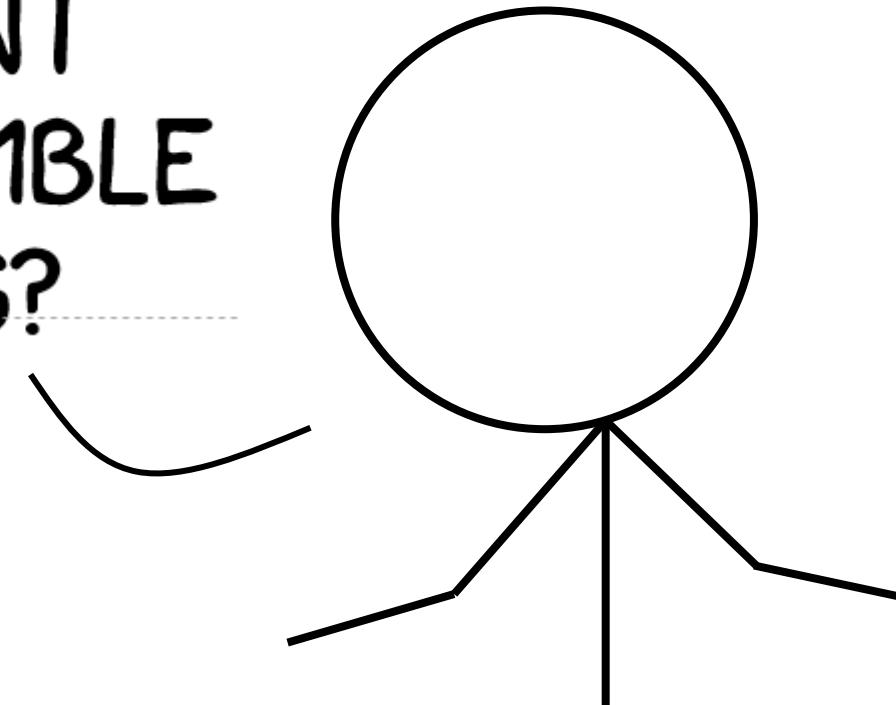
Name	Address
_init	400460
sub_400500	400500
puts	400510
printf	400520
read	400530
__libc_start_main	400540
strcmp	400550
open	400560
exit	400570
_start	400580
sub_4005a9	400590
call_gmon_start	4005ac
_do_global_dtors_aux	4005cd
frame_dummy	400640
authenticate	400664
accepted	40066d
rejected	4006f5
main	40071d
__libc_csu_init	4007e0
__libc_csu_fini	400870
_do_global_dtors_aux	400888
_fini	400888
puts	1000000
read	1000010
exit	1000020
printf	1000030
__libc_start_main	1000040

The right pane shows the assembly code for the main function:

```
main:
    .push rbp
    .mov rbp, rsp
    .sub rsp, 0x40
    .mov dword ptr [rbp-0x34], edi
    .mov qword ptr [rbp-0x40], rsi
    .mov byte ptr [rbp-0x8], 0x0
    .mov byte ptr [rbp-0x10], 0x0
    .mov edi, 0x400915 "Username: "
    .call rbp
    .mov rax, [rbp-0x10]
    .mov rdx, 0x8
    .mov rsi, rax
    .mov edi, 0x0
    .call rbp
    .mov rax, [rbp-0x24]
    .mov rdx, 0x1
    .mov rsi, rax
    .mov edi, 0x0
    .call rbp
    .mov rax, [rbp-0x24]
    .mov rdx, 0x1
    .mov rsi, rax
    .mov edi, 0x0
    .call rbp
    .lea rax, [rbp-0x20]
    .mov rdx, 0x8
    .mov rsi, rax
    .mov edi, 0x0
    .call rbp
    .mov rax, [rbp-0x24]
    .mov rdx, 0x1
    .mov rsi, rax
    .mov edi, 0x0
    .call rbp
    .lea rdx, [rbp-0x20]
```

Patching support
in
angr Management

HOW CAN I
REASSEMBLE
BINARIES?



Limitations

- The **infeasibility** of static content classification
- The lack of guarantee of our approaches
- The “80% versus 20%” problem

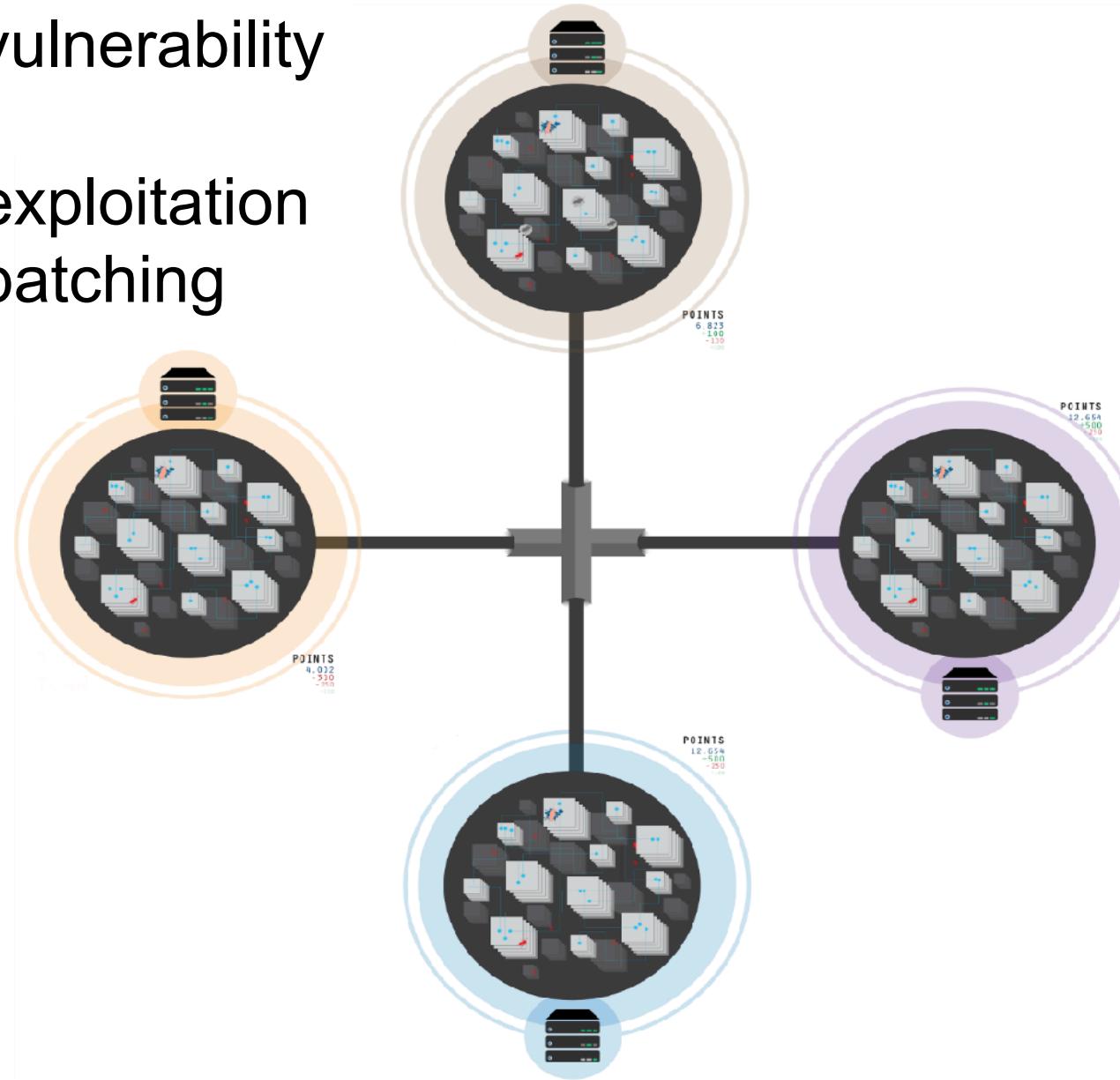
Brief Results: Success Rate (cont.)

- Emphasis

We reproduced Uroboros' results on Coreutils 8.15 compiled with GCC 4.6 on Ubuntu 12.04

- Changes in Coreutils > 8.15 makes it harder for Uroboros
- Optimizations in GCC 5 yields new challenges for Uroboros

- Autonomous vulnerability discovery
- Autonomous exploitation
- Autonomous patching



- Autonomous vulnerability discovery
- Autonomous exploitation
- Autonomous **patching**

Requires
a low memory overhead
and
an **EXTREMELY** low execution overhead